

Operating System/2 Programming Tools and Information Version 1.2/1.3

I/O Subsystems and Device Support Volume 1 and Volume 2 Technical Upgrade

91F8684

Operating System/2
Programming Tools
and Information
Version 1.2/1.3

I/O Subsystems and Device Support
Volume 1 and Volume 2
Technical Upgrade

First Edition (December 1990)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

COPYRIGHT LICENSE: This publication contains printed sample application programs in source language, which illustrate OS/2 programming techniques. You may copy and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the OS/2 application programming interface.

Each copy of any portion of these sample programs or any derivative work, which is distributed to others, must include a copyright notice as follows: "© (your company name) (year) All Rights Reserved."

© Copyright International Business Machines Corporation 1990. All rights reserved.
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Special Notices

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM
IBM C/2
IBM COBOL/2
IBM FORTRAN/2
IBM Macro Assembler/2
Operating System/2
OS/2
Presentation Manager

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

Microsoft Corporation

© Copyright IBM Corp. 1990

About This Book

This book, as well as the other books in the upgrade package, contains programming information that is new to or different from the information in the *IBM* Operating System/2* (OS/2*) Programming Tools and Information Version 1.2.* library (part number 6024929). In addition, information from the *OS/2 Programming Tools and Information Version 1.2 Technical Update* (part number 64F1705) is included in this upgrade package.

The upgrade describes features added by the OS/2 Version 1.3 product, and amends some of the information that was published with OS/2 Version 1.2.

How To Use This Book

This book contains a section for each of the following books:

Book Title	***Book Number***
I/O Subsystems and Device Support: Volume 1 Device Drivers	64F0282
I/O Subsystems and Device Support: Volume 2 Presentation Driver Interfaces	64F0283

Only those chapters of the Version 1.2 book that are changed by the Version 1.3 product are included herein. The chapter numbers and titles are the same as those in the Version 1.2 book. As a convenience, the beginning of each chapter in this book provides a summary of changes for that chapter.

Who Should Read This Book

This book is for application designers and programmers who are using the components of the *IBM OS/2 Version 1.2/1.3 Programming Tools and Information Technical Upgrade* package. The reader is assumed to be familiar with the services of OS/2.

* Trademark of the IBM Corporation

© Copyright IBM Corp. 1990

Contents

I/O Subsystems and Device Support: Volume 1 1

Chapter 2. Device Driver Overview 3
Device Driver INIT-Time Function Call Summary 3

Chapter 4. Device-Driver Strategy Commands 5
Request Packets 5

Chapter 5. Device Helper Services 7
DevHlp Services and Function Codes 7
DevHlp Services and Device Contexts 7
DispMsg Display Message 7
MonWrite Give Data to Monitors 8
VideoPause Suspend/Resume Video Active Threads 9

Chapter 6. Device Drivers 11
Mouse Device Overview 11
Inter-Device-Driver Communication (IDC) Interface Descriptions 11
MOUSE$ IDC Interface 12
Device-Dependent Device Driver IDC Interface 14
Keystroke Monitors 15
The Printer Device Driver 16
Font Monitor Packet 16
Print-Monitor-Job Title Packet 16
Monitor-Job Title Packet 17

Chapter 7. Generic IOCtl Commands 19
Generic IOCtl Examples 19
Category 1 — Function 53H 20
Category 1 — Function 73H 20
Category 4 — Function 50H 20
Category 4 — Function 56H 20
Category 5 — Function 4DH 21
Category 5 — Function 4EH 21
Category 5 — Function 6EH 22
Category 8 — Function 43H 23
Category 8 — Function 63H 23

Chapter 8. Character Device Monitors 25
Monitoring Character Device Data Streams 25
Device Monitor Support Limitations 25

I/O Subsystems and Device Support: Volume 2 27

Chapter 12. Mandatory Functions for All Devices 29
GreQueryLogColorTable 29
GreCreateLogColorTable 29
GreEscape, DEVESC_GETCP 29
GreEscape, DEVESC_SETMODE 29

© Copyright IBM Corp. 1990

GreEscape, DEVESC_DRAFTMODE 30

Chapter 13. Mandatory Functions for Display Devices 31
GreDeviceSetAVIOFont 31
GreDeviceSetDCOrigin 31

Chapter 14. Simulated Functions 33
GreSetupDC 33

Chapter 15. Graphics Engine Internal Functions 35
GreGetAttributes 35

Chapter 17. Presentation Manager Spooler 37
The Queue Processor (Queue Driver) 37
How a Queue Processor Prints 37
SplQpPrint 38
SplQpQueryDt 38

Master Index 39

Index 41

I/O Subsystems and Device Support: Volume 1

© Copyright IBM Corp. 1990

Chapter 2. Device Driver Overview

Summary of Changes

New	Updated	Section Title
	√	Device Driver INIT-Time Function Call Summary

Device Driver INIT-Time Function Call Summary

On page 2-8, add the following:

DosGetInfoSeg Get Address of System Variables Segment

© Copyright IBM Corp. 1990

Chapter 4. Device-Driver Strategy Commands

Summary of Changes

New	Updated	Section Title
	√	Request Packets

Request Packets

On page 4-3, add the following error code to Table 4-3 (Status-Field Error Codes):

Error Codes	Description
14H	Device already in use.

© Copyright IBM Corp. 1990

Chapter 5. Device Helper Services

Summary of Changes

New	Updated	Section Title
	√	DevHlp Services and Function Codes
	√	DevHlp Services and Device Contexts
√		DispMsg
	√	MonWrite
√		VideoPause

DevHlp Services and Function Codes

On page 5-2, in the table of DevHlp Services and Function Codes, change code **38** to **3A**, and add the following services:

```
VideoPause  3C     Suspend/resume video active threads
DispMsg     3D     Display message (for base device drivers)
```

DevHlp Services and Device Contexts

On page 5-4, in the table of DevHlp Service Modes, change code **38** to **3A**, and add the following services:

DevHlp Service	Code	Kernel	Interrupt	User	Init
VideoPause	3C	*	*		*
DispMsg	3D				*

On page 5-5, add the DispMsg and VideoPause services to **10. System Services**.

DispMsg
Display Message

This function displays a message from a base device driver on the system console.

Processing

```
MOV  ES,SEG  Msg_Table     ; ES = segment of message table
MOV  DI,OFFSET  Msg_Table  ; DI = offset of message table
XOR  BX,BX                 ; BX = Reserved (must be 0)
MOV  DL,DevHlp_DispMsg
CALL [Device_Help]
```

Results

None.

© Copyright IBM Corp. 1990

Remarks

The message is not displayed immediately but is queued until system initialization retrieves it from the system message file.

The structure of the message table is:

```
WORD     Message ID
WORD     Number of fill-in items
DWORD    Pointer to first fill-in item of ASCIIZ string
DWORD    Pointer to second fill-in item of ASCIIZ string
 .
 .
 .
DWORD    Pointer to last fill-in item of ASCIIZ string
```

If an error message is displayed, and *pausenoerror=no* is not contained in config.sys, the user is prompted to **press any key to continue.**

MonWrite
Give Data to Monitors

On page 5-29, replace the contents of the entire page with the following:

Purpose

MonWrite passes data records to the monitors for filtering.

Processing

```
LDS  SI,data_record_offset     ; Offset of data
record in DS
MOV  CX,count                  ; Byte count of data record
MOV  AX,monitor_handle         ; Handle for chain
returned from
                               ; Previous MonitorCreate call
MOV  DI,millisecond_high       ; High word of milliseconds
MOV  BX,millisecond_low        ; Low word of milliseconds
MOV  DH,wait_flag              ; Wait/Nowait/Wait_timeout flag MOV DL,DevHlp_MonWrite
CALL [Device_Help]
```

Results

```
'C' Clear if OK
    AX = 0
'C' Set if error
    AX = error code
         Invalid monitor handle
         Not enough memory
         Call interrupted
         Error semaphore timeout
```

Remarks

This function may be called at task time or interrupt time, in either the OS/2 mode or DOS mode. Wait_flag is set to 0 if the MonWrite request occurs at task or user time and the device driver indicates that the monitor dispatcher do the synchronization. That is, if wait_flag is set to 0, the device driver waits until the data can be placed into the monitor chain before the monitor dispatcher returns to the device driver. If wait_flag is set to 1, the device driver does not wait; and if the data cannot be placed into the monitor chain, the monitor dispatcher will return immediately with the appropriate error. Wait_flag must be set to 1 if the MonWrite request occurs at

interrupt time. Wait_flag is set to 2 if the MonWrite request occurs at task or user time, and the device driver indicates that the monitor dispatcher do the synchronization for the time in milliseconds specified in DI:BX.

The DS register must be set to the device-driver data segment.

The error, NOT ENOUGH MEMORY, will be returned to the device driver when the MonWrite call is made and the monitors are not able to receive the data. If this condition occurs at interrupt time, an overrun occurred. If it occurs at task (or user) time, the process can block.

The error, NOT ENOUGH MEMORY, also will be returned to the device driver when a flush record, sent to the monitors by a previous MonFlush call, was not returned to the device driver.

If the thread on which the device driver calls MonWrite blocks (the device driver specified the wait option) and is awakened because the process that owns the thread is terminating, a call-interrupted error is returned to the device driver. The device driver must return the error to the caller so that the process can complete termination.

Each call to MonWrite will send a single complete record. The data sent by this call is considered to be a complete record. A data record must not be longer than two bytes less than the length of the device driver's monitor chain buffer.

The state of the interrupt flag is not preserved across calls to DevHlp.

See Chapter 8, "Character Device Monitors," for more information about the use of this function.

VideoPause
Suspend/Resume Video Active Threads

This function is called by device drivers when the controller reports a DMA overrun. VideoPause starts or stops high-priority threads. This halts threads using the CPU for video transfers, which allows the diskette DMA to complete termination properly.

Processing

```
MOV  AL,on_or_off          ; VideoPause flag
                           ; = 0     off
                           ; < > 0   on
MOV  DL,DevHlp_VideoPause
CALL [Device_Help]
```

Results

None.

Remarks

Use this function after a retry has failed. Turn VideoPause on just long enough to accomplish the transfer; otherwise, impairment of the system could occur. If multiple device drivers turn VideoPause on, it is not turned off until all device drivers have turned it off.

Chapter 6. Device Drivers

Summary of Changes

New	Updated	Section Title
	√	Mouse Device Overview
	√	Inter-Device Drive Communication (IDC) Interface Description
	√	Keystroke Monitors
	√	The Printer Device Driver
	√	Font Monitor Packet
√		Print-Monitor-Job Title Packet
√		Monitor-Job Title Packet

Mouse Device Overview

Note: Throughout this section, two terms, *mouse* and *MOUSE$*, are used repeatedly. They are defined as follows:

- Mouse: Any class of pointing device, unless otherwise noted.
- MOUSE$: A device-independent device driver.

On page 6-37, replace the first paragraph with the following:

The OS/2 Operating System provides support for a pointing device for both DOS mode and OS/2 mode applications. Two classes of pointing devices are supported: relative and absolute. A relative pointing device reports relative motion (how far it has moved). An example of a relative pointing device is a mouse or a track ball. An absolute pointing device reports absolute positions within some predefined device space. With an absolute pointing device, there is no real concept of relative motion. An example of an absolute pointing device is a touch-sensitive screen. The following components provide pointing-device support:

- A mouse device driver
- A mouse router
- A base mouse subsystem
- A pointer draw device driver.

Inter-Device-Driver Communication (IDC) Interface Descriptions

Starting on page 6-41, replace the IDC interface descriptions from the "Device-Independent Mouse Device IDC Description" up to (but not including) "OS/2 Mode Mouse Support" at the top of page 6-44, with the following:

This section describes the supported device-independent and dependent device driver IDC interfaces. For all IDC requests, follow these steps:

1. Load AX with the request number.
2. Initialize required data areas.
3. Load other required registers.

© Copyright IBM Corp. 1990

4. Load DS with the data segment/selector of the called device driver.
5. Load ES with the data segment/selector of the calling device driver.
6. Execute a far call to the IDC entry point in the other device driver.
7. Examine the carry flag when the far call is returned. **Carry set** indicates that the request failed. **Carry clear** indicates that the request was processed.

All registers not explicitly listed contain undefined values. Register values are not preserved across IDC requests, with exception to values for segment registers. The state of the interrupt flag **must** be preserved across all IDC requests.

Use caution when loading DS and ES, and when executing the far call. These operations are processor-mode dependent. **AttachDD** will return both real- and protect-mode entry points, as well as segment and selector values for the data segment of the attached-to device driver.

MOUSE$ IDC Interface

The IDC provided by MOUSE$ supports the following:

- Process_Packet
- Disable_Support
- Process_Absolute.

Process_Packet

This request is issued by the device-dependent device driver to process a relative-pointing-device event, usually from a mouse device. Before requesting Process_Packet, issue an end-of-interrupt (EOI), and place the pointing device in a disabled state. For a description of a disabled state, see "Disable_Device" on page 15 in this book. When necessary, use MOUSE$ to return the device to an enabled state. Process_Packet is issued only when the pointing device is enabled. For a discussion of interrupt processing, see "Interrupt Processing" on page 6-40 of the *OS/2 Version 1.2 I/O Subsystems and Device Support, Volume 1 Device Drivers* book.

Input:

AX = Process_Packet function code, 0001H.
Common-event buffer initialized.

The offset of the common-event buffer was passed on the Read_Enable request. This data buffer is used to pass both relative and absolute pointing-device events. When the device driver issues a Process_Packet request, the buffer has the following format:

```
Event_Data   Struc

Event        dw     ?   ; Description of event.
Col_Motion   dw     ?   ; Relative column motion.
Row_Motion   dw     ?   ; Relative row motion.

Event_Data   Ends
```

Before the device driver makes the request, initialize the buffer. Note that positive motion is up and to the right. Up can also be viewed as moving a mouse device away from the user. Negative motion is the opposite of positive.

Output: No specific output for request.

Disable_Support

This request is used by the device-dependent device driver to inform MOUSE$ that it has *DeInstalled*. When the request is received, MOUSE$ enters a disabled state. It returns the error, DEVICE NOT READY, for subsequent requests from the system. Do **not** issue this request under normal circumstances.

Input:

AX = Disable_Support function code, 0002H.

Output: No specific output for the request.

Process_Absolute

This request is issued by the device-dependent device driver to process an absolute-pointing device event, usually from a touch-sensitive screen. Before requesting Process_Absolute, issue an EOI, and place the pointing device in a disabled state. For a description of a disabled state, see "Disable_Device" on page 15 in this book. When necessary, use MOUSE$ to return the device to an enabled state. Process_Absolute is issued only when the pointing device is enabled. For a discussion of interrupt processing, see "Interrupt Processing" on page 6-40 of the *OS/2 Version 1.2 I/O Subsystems and Device Support, Volume 1 Device Drivers* book. The reported event is mapped into the current display mode and appears to the system to be a mouse event.

Input:

AX = Process_Absolute function code, 0003H.
Common-event buffer initialized.

The offset of the common-event buffer was passed on the Read_Enable request. This data buffer is used to pass both relative and absolute pointing-device events. When issuing a Process_Absolute request, the buffer has the following format:

```
Event_Data   Struc

Event        dw     ?   ; Description of event.
Row_Pos      dw     ?   ; Row position of event.
Col_Pos      dw     ?   ; Col position of event.
Row_Size     dw     ?   ; Number of row units.
Col_Size     dw     ?   ; Number of column units.

Event_Data   Ends
```

MOUSE$ uses Row_Size and Col_Size to map the absolute event into the current display mode. These values should reflect the maximum allowed position that could be reported. All values are zero-based. The **event** field should never indicate that motion was associated with the event. MOUSE$ will determine if motion occurred. The upper-left corner of the device should be reported as location 0,0.

Initialize the buffer before making the request.

Output: No specific output for the request.

Device-Dependent Device Driver IDC Interface

The IDC provided by the device-dependent device driver must support the following requests:

- Query_Config
- Read_Enable
- Read_Disable
- Enable_Device
- Disable_Device.

Query_Config

This request is issued by MOUSE$ and determines the operating characteristics of the attached hardware. The format follows.

Input:

AX = Query_Config function code, 0001H.
DI = Offset in mouse data segment of where to write the configuration data. Note that ES was loaded with the MOUSE$ data segment/selector prior to the request. The configuration data has the following format:

```
Config_Data  Struc

Length       dw    ?  ; Total length in bytes (6)
Num_Mics     db    ?  ; Device resolution in mickeys/centimeter.
Num_Butt     db    ?  ; Number of buttons (up to 7 supported)
Dev_IRQ      db    ?  ; Device IRQ level
Mouse_Type   db    ?  ; Type of mouse attached:
                      ;   0 = Unknown
                      ;   1 = Bus mouse
                      ;   2 = Serial mouse
                      ;   3 = Inport mouse
                      ;   4 = PS/2 inboard
                      ;   5-255 = Reserved
Config_Data  Ends
```

Initialize the **Length** field to 6 before issuing the request. An absolute pointing device, such as a touch-sensitive screen, reports a mouse type of zero (unknown). The **Num_Mics** field for an absolute device should reflect some reasonable measure of the number of device units per centimeter.

Output: Configuration data structure filled if no errors.

Read_Enable

This request serves two purposes. It gives the offset of the common-event buffer in the MOUSE$ data segment. This offset should be saved. It also informs the device-dependent device driver that it can start sending event data to MOUSE$ using the Process_Packet and Process_Absolute requests. The device-dependent device driver also should attach itself to MOUSE$, using the Device Helper function AttachDD, when it receives this request.

Input:

AX = Read_Enable function code, 0002H.
DI = Offset to common-event buffer.

The common-event buffer resides in the MOUSE$ data segment. To address it, use the MOUSE$ data segment/selector value, depending on the processor mode, returned on the AttachDD call. The

common-event buffer has enough space to format either a relative or absolute event. For the two common-event buffer formats, see "Process_Packet" on page 12, and "Process_Absolute" on page 13 in this book.

Output: No specific output for the request.

Read_Disable

This request is issued by MOUSE$ when it no longer accepts Process_Packet or Process_Absolute requests. References to the common-event buffer should be discontinued. However, MOUSE$ may enable the interface again by sending another Read_Enable request. Normally this request is not issued.

Input:

AX = Read_Disable function code, 0003H

Output: No specific output for the request.

Enable_Device

This request is used by MOUSE$ when resetting a pointing device to an enabled state. Process_Packet and Process_Absolute requests are made to MOUSE$ when a pointing device is in an enabled state. Receipt of an Enable_Device request is not considered an error if the pointing device is already enabled.

Input:

AX = Enable_Device function code, 0004H.

Output: No specific output for the request.

Disable_Device

This request is used by MOUSE$ when placing a pointing device in a disabled state. Upon completion of this request, no Process_Packet or Process_Absolute requests are to be made to MOUSE$ until an Enable_Device request is received. There is no requirement that the hardware itself be disabled; only that no Process_Packet or Process_Absolute requests be issued. Receipt of a Disable_Device request is not considered an error if the pointing device is already disabled.

Input:

AX = Disable_Device function code, 0005H.

Output: No specific output for the request.

Keystroke Monitors

An application operating with OS/2 Version 1.2 and Version 1.3 cannot register a keyboard monitor for a Presentation Manager session. The following information clarifies this restriction.

On page 6-66, replace the second paragraph under **Keystroke Monitors** with the following:

When DosMonReg is issued with keyboard devices, the INDEX parameter indicates which OS/2 session the caller requested the monitor to be registered on. Note that the INDEX parameter indicates an OS/2 session from 0 to 15 (see DosGetInfoSeg).

An INDEX parameter value of -1 indicates that the session of the calling thread is being requested. If the caller requests either the Presentation Manager session or

the DOS mode session, a MONITORS NOT SUPPORTED error is returned. For both of these sessions, the error return code will be generated, regardless of the caller's specific usage of the INDEX parameter (explicit/implicit).

The Printer Device Driver

On page 6-128, replace the table (under the first paragraph) with the following:

PRINT01.SYS (part of Basedd01.sys)	supporting parallel port adapters on Family One bus machines.
PRINT02.SYS (part of Basedd02.sys	supporting parallel port adapters on machines containing Micro Channel Architecture.

Font Monitor Packet

On page 6-143, replace the table (under the first paragraph) with the following:

Table 6-24. Monitor Font Packet	
System File Number	WORD
Command Byte	BYTE
Reserved, Set to 0	BYTE
Reserved, Set to 0	BYTE
Reserved, Set to 0	BYTE

Print-Monitor-Job Title Packet

On page 6-146, (after "Monitor Status Packet") add the following new information:

A print-monitor-job title packet passes the name of the file being printed to all monitors registered with the printer device driver. The flags are defined in Figure 1 on page 17.

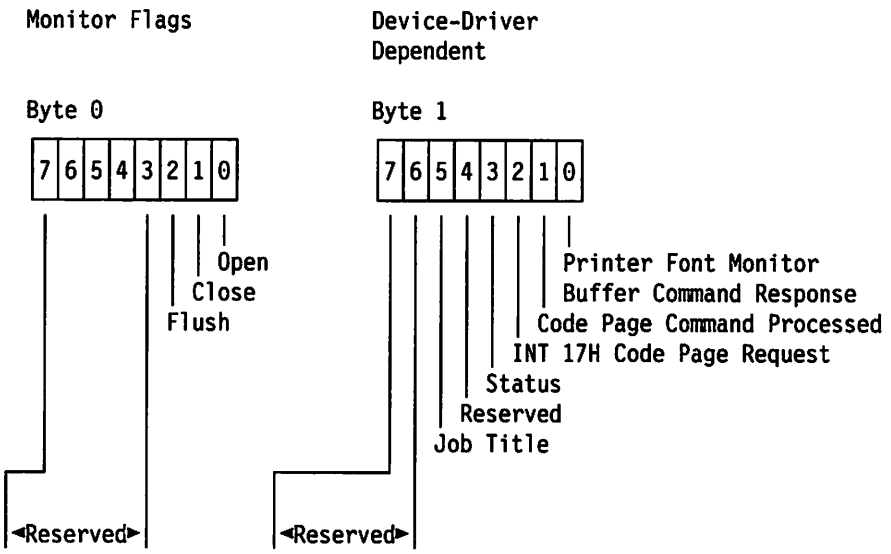


Figure 1. Monitor Packet Flags

Monitor-Job Title Packet

When the job title bit is set to 1 (byte 1, bit 5), it indicates the monitor buffer is a job title packet from the printer device driver. In this case, the next bytes are:

Monitor-Job Title Packet	
System File Number	WORD
Job Title Length	WORD
Job Title Buffer	BYTES

The job title length field includes the null character. The job title buffer is terminated with the null character.

The job title buffer field contains byte 6 through byte 131.

Chapter 7. Generic IOCtl Commands

Summary of Changes

New	Updated	Section Title
√		Generic IOCtl Examples
	√	Category 1, Function 53H
	√	Category 1, Function 73H
	√	Category 4, Function 50H
	√	Category 4, Function 56H
√		Category 5, Function 4DH
√		Category 5, Function 4EH
√		Category 5, Function 6EH
	√	Category 8, Function 43H
	√	Category 8, Function 63H

Generic IOCtl Examples

On page 7-3, add the following to the list of categories and functions for the GENERIC IOCtl request:

CAT	FUNCTION	DESCRIPTION
05	47H	Reserved
	4BH	Reserved
	4CH	Reserved
	4DH	Set Printer Job Title
	4EH	Set Printer Timeout Value
	4FH	Reserved
	67H	Reserved
	6BH	Reserved
	6CH	Reserved
	6DH	Reserved
	6EH	Return Printer Timeout Value
	6FH	Reserved

© Copyright IBM Corp. 1990

Category 1 — Function 53H

On page 7-17, change the **Flags3** settings to the following:

Note	Bit	Meaning
10	3-4	Extended Hardware Buffering

Bit 4	*Bit 3*	
0	0	Not Supported; ignored. No change to FIFO state.
0	1	Extended Hardware Buffering DISABLED
1	0	Extended Hardware Buffering ENABLED
1	1	Automatic Protocol Override

Category 1 — Function 73H

On page 7-44, change the definition of **Flags3, bits 3-4** to the following:

Extended Hardware Buffering

Bit 4	*Bit 3*	
0	0	Not supported
0	1	Extended Hardware Buffering DISABLED
1	0	Extended Hardware Buffering ENABLED
1	1	Automatic Protocol Override

Category 4 — Function 50H

On page 7-56, replace the **Parameter Packet Format** table with the following:

Field	Length
Code Page Translation Table	OTHER

Category 4 — Function 56H

On page 7-72, under **Remarks** add the following:

When hot key ID FFFFh is set, Alt+Ctrl+Del, Alt+Esc, and Alt+Ctrl+NumLock keys are disabled. You can reinstate these keys by reissuing IOCtl 56H (Set Session Manager Hot Key).

Category 5 — Function 4DH

Purpose

Set print-job title.

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Length	WORD
Address of ASCIIZ job title	DWORD

Where:

Command Information
is reserved and must be set to zero.

Length
is the length of the buffer containing the print-job title. This includes the terminating null character, but excludes the length field.

Address of Job Title
is the address of the ASCIIZ string containing the print-job title. A null character terminates the ASCIIZ string.

Returns

in the kernel request-packet status field:

0100H Completed successfully.
8113H Invalid parameter if the address of data packet is invalid.
8113H Invalid parameter if the address of the ASCIIZ string is invalid.

Remarks

If the ASCII string has more than 126 bytes, the printer device driver truncates it to the first 125 bytes, plus 1 byte for the null character.

Category 5 — Function 4EH

Purpose

Set printer-timeout value.

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Timeout Value in Seconds	WORD

Where:

Command Information
is reserved and must be set at zero.

Timeout Value
is the length of time, in seconds, that the printer device driver waits for a hardware interrupt to occur.

Returns

in the kernel request-packet status field.

0100H Completed successfully
8113H Invalid parameter if the data-packet address is invalid.

Remarks

This IOCtl informs the base, physical, printer device driver of the length of time (in seconds) to wait for a hardware interrupt before a print request is canceled.

Valid timeout values range between 0 and 65535 seconds. An application setting this value must also issue PRFWriteProfileString to update the OS2SYS.INI file.

Category 5 — Function 6EH

Purpose

Return printer timeout value.

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Timeout Value in Seconds	WORD

Where:

Command Information
is reserved and must be set to zero.

Timeout Value
is the length of time, in seconds, that the printer device driver waits for a hardware interrupt to occur.

Returns

in the kernel request status field:

0100H	Completed successfully
8113H	Invalid parameter if the data-packet address is invalid.

Remarks

Valid timeout values range between 0 and 65535 seconds.

Category 8 — Function 43H

Purpose

On page 7-143, under **Device type**, add the following:

Value	**Meaning**
8	R/W optical disk

On page 7-143, under **Device attributes**, change the device attribute bits to the following:

Bit	**Description**
0	Removable media flag. If this bit is set to 1, the media cannot be removed. If this bit is set to 0, the media is removable.
1	Changeline flag. If this bit is set to 1, the device support can determine if the media was removed since the last I/O operation. Query whether the media has changed by calling the device driver strategy command 1H/Media Check. If this bit is set to 0, then the device driver can return the value 0 (unsure if media has changed) from the Media Check call.

Category 8 — Function 63H

Purpose

On page 7-148, under **Device type**, add the following:

Value	**Meaning**
8	R/W optical disk

On page 7-149, under **Device attributes**, change the device attribute bits to the following:

Bit	Description
0	Removable media flag. If this bit is set to 1, the media cannot be removed. If this bit is set to 0, the media is removable.
1	Changeline flag. If this bit is set to 1, the device support can determine if the media was removed since the last I/O operation. Query whether the media has changed by calling the device driver strategy command 1H/Media Check. If this bit is set to 0, then the device driver can return the value 0 (unsure if media has changed) from the Media Check call.

Chapter 8. Character Device Monitors

Summary of Changes

New	Updated	Section Title
	√	Monitoring Character Device Data Streams
	√	Device Monitor Support Limitations

Monitoring Character Device Data Streams

OS/2 Version 1.2 and Version 1.3 do not allow an application to register a keyboard or mouse device monitor for a Presentation Manager session. The following information documents this restriction.

On page 8-2, replace the second paragraph under **Monitoring Character Device Data Streams** with the following:

The flow of data between a device driver and its device is called a *data stream*. The device driver defines the data streams for its device and associates the data it receives from a device or application with a particular data stream. Device drivers may have more than one data-stream model within or between sessions. For example, the keyboard device driver supports three different data-stream models for DOS mode, Presentation Manager, and OS/2 mode sessions. Of these, only the OS/2 mode data stream supports keystroke monitors.

On page 8-2, replace the fifth paragraph under **Monitoring Character Device Data Streams** with the following:

Just as device drivers define their data streams, they also determine how character monitors are chained. The keyboard and mouse device drivers, for example, support a monitor chain for each OS/2 mode session. The printer device driver supports two monitor chains for each physical printer device. See "Mouse Device Driver", "Console Device Drivers (Screen and Keyboard)", and "The Printer Device Driver" sections in the original *OS/2 Version 1.2 I/O Subsystems and Device Support, Volume 1 Device Drivers* book for a description of how these device drivers support device monitors.

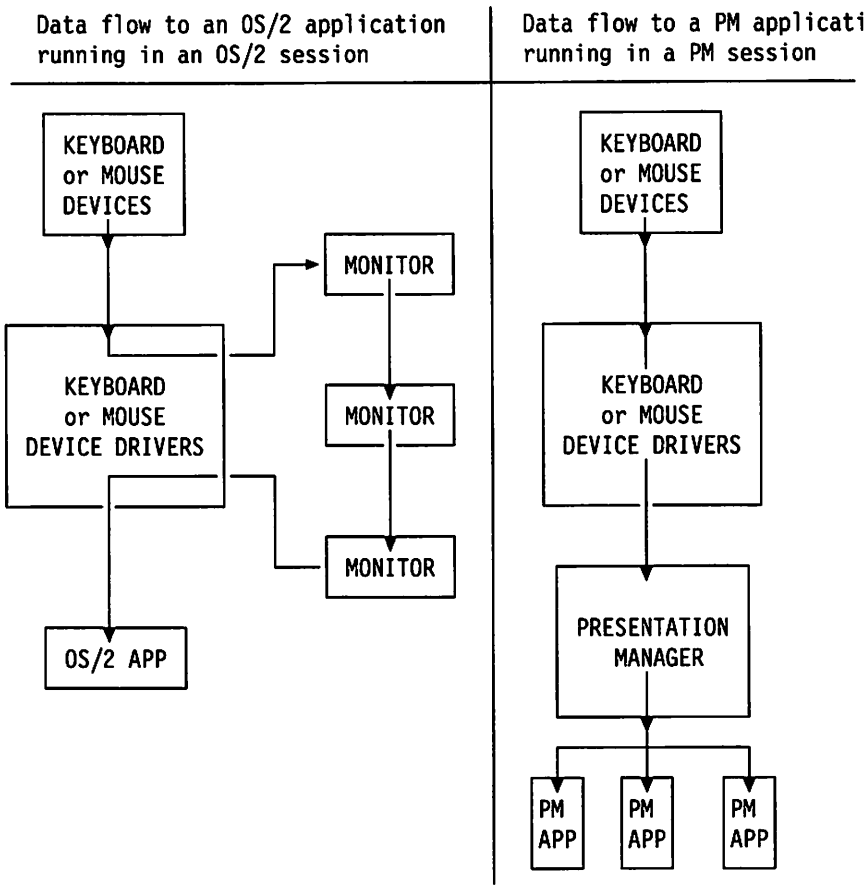
Device Monitor Support Limitations

On page 8-3, replace the first paragraph under **Device Monitor Support Limitations** with the following:

Data streams for character devices may be monitored by a character-device monitor, only when the character device driver provides device monitor support. Applications, including both OS/2 mode and Presentation Manager applications, may monitor keystrokes and mouse clicks for all OS/2 mode sessions. However, the applications may not monitor keystrokes and mouse clicks for the Presentation Manager session.

© Copyright IBM Corp. 1990

On page 8-4, replace Figure 8-3, **OS/2 Monitors and PM Applications**, as follows:



Remember that applications may no longer intercept keystrokes or mouse clicks associated with the Presentation Manager session through keystroke or mouse monitors.

© Copyright IBM Corp. 1990

Chapter 12. Mandatory Functions for All Devices

Summary of Changes

New	Updated	Section Title
	√	GreQueryLogColorTable
	√	GreCreateLogColorTable
	√	GreEscape, DEVESC_GETCP
	√	GreEscape, DEVESC_SETMODE
	√	GreEscape, DEVESC_DRAFTMODE

GreQueryLogColorTable

On page 12-95, change CLR_NOINDEX in the description of the **pArray** parameter to QLCT_NOTLOADED.

GreCreateLogColorTable

On page 12-96, add the following to the options listed under **flOptions**:

LCOL_SYSCOLOR This flag is set when the color data addressed by **pData** is for the system color table. For details see WinSetSysColors in the *Programming Reference*.

GreEscape, DEVESC_GETCP

On page 12-123, delete all information about the DEVESC_GETCP subfunction of GreEscape. This escape code is not used and presentation drivers should not provide support for it.

GreEscape, DEVESC_SETMODE

In the GreEscape section of Chapter 12, add the following description of the DEVESC_SETMODE subfunction:

The DEVESC_SETMODE escape code requests the presentation driver to set a specific mode of the output device.

DEVESC_SETMODE is currently defined only for code-page mode (mode=0). The action taken by the handling routine in the presentation driver depends upon what support the device provides for multiple code pages:

1. If the device has the requested code page built into it, the handling routine issues the controls to select the code page.
2. If the device supports downloaded code pages, the handling routine issues the calls and controls needed to load and select the requested code page.

Note: DEVESC_SETMODE is metafiled and is recorded by the spooler.

© Copyright IBM Corp. 1990

Stack Frame

hdc Device-context handle.

lEscape DEVESC_SETMODE

clnCount Number of bytes pointed to by plnData.

plnData Pointer to an ESCMODE structure:

ulMode Identifies the mode that is to be changed. Defined values for ulMode are:

0 Code page.

abModedata The format and meaning of abModedata depends upon the value of ulMode. When ulMode=0, abModedata contains the requested code page number expressed as a USHORT value.

pcOutCount The handling routine should ignore this parameter.

pOutData The handling routine should ignore this parameter.

Return Codes

The handling routine returns:

DEV_OK	Successful.
DEVESC_NOTIMPLEMENTED	This escape code is not implemented.
DEVESC_ERROR	Error.

GreEscape, DEVESC_DRAFTMODE

On page 12-120, add the following to the description of the DEVESC_DRAFTMODE subfunction:

Note that the driver's response to a call to GreDeviceQueryFonts is influenced by the current setting, off or on, of draft mode. If the device supports fonts that cannot be positioned to the nearest pel, the GreDeviceQueryFonts routine in the driver should only return details of these fonts when draft mode is on.

Chapter 13. Mandatory Functions for Display Devices

Summary of Changes

New	Updated	Section Title
	√	GreDeviceSetAVIOFont
	√	GreDeviceSetDCOrigin

GreDeviceSetAVIOFont

On page 13-9, the name, function number, and parameters of GreDeviceSetAVIOFont have been changed to allow the graphics engine to pass logical font information to the presentation driver. The function name is changed to GreDeviceSetAVIOFont2. Also, the Presentation Manager's graphics engine has been modified so that calls to load or delete an image font for an AVIO presentation space are passed through the dispatch table to the handling routine for GreDeviceSetAVIOFont2. To minimize the impact on existing drivers, the default handling routine for GreDeviceSetAVIOFont2 removes the logical font information and redirects the call through the dispatch table to GreDeviceSetAVIOFont.

The syntax for GreDeviceSetAVIOFont2 is:

```
(BOOL) GreDeviceSetAVIOFont2 (hdc, pFat, pFontDef,
                              lcid, pInstance, lFunction)
```

Differences between GreDeviceSetAVIOFont2 and GreDeviceSetAVIOFont are shown below. See GreDeviceSetAVIOFont on page 13-9 for a full description.

pFat *(PFATTRS)*
Pointer to a font attributes structure for the specified font.

lFunction *(ULONG)*
The symbolic name NGreDeviceSetAVIOFont2 identifies the function number in the low-order word of **lFunction**.

GreDeviceSetDCOrigin

On page 13-22, change the paragraph under *Support* to:

This function must be supported by presentation drivers for display devices. Drivers for other device types must not include support for GreDeviceSetDCOrigin: these drivers must allow calls to this function to be directed, through the dispatch table, to the default handling routine.

© Copyright IBM Corp. 1990

Chapter 14. Simulated Functions

Summary of Changes

New	Updated	Section Title
	√	GreSetupDC

GreSetupDC

On page 14-77, change the paragraph under *Support* to:

This function is supported by the graphics engine and can be hooked by presentation drivers for display devices. Drivers for other device types must not hook GreSetupDC.

© Copyright IBM Corp. 1990

Chapter 15. Graphics Engine Internal Functions

Summary of Changes

New	Updated	Section Title
	√	GreGetAttributes

GreGetAttributes

On page 15-24, note that this function can also return PMERR_UNSUPPORTED_ATTR.

© Copyright IBM Corp. 1990

Chapter 17. Presentation Manager Spooler

Summary of Changes

New	Updated	Section Title
	√	The Queue Processor (Queue Driver)
	√	SplQpPrint
	√	SplQpQueryDt

The Queue Processor (Queue Driver)

On page 17-20, replace the description of the Queue Processor (Queue Driver) with the following:

Note: The Presentation Manager shell uses the term *Queue Driver* to identify the queue processor.

A queue processor takes a spool file and prints it. Each queue has its own queue processor. The Presentation Manager system queue processor is supplied in the file PMPRINT.QPR.

The following functions provide an interface to the queue processor:

SplQpClose
SplQpControl
SplQpInstall
SplQpOpen
SplQpPrint
SplQpQueryDt.

In addition to these functions, a visual interface is supplied through the SplMessageBox entry point.

You may supply queue processors of your own to support user data types, but any queue processor that you create must support the PM_Q_STD and PM_Q_RAW standard data types, also its file name must have the .QPR extension, and you must export the functions listed above.

How a Queue Processor Prints

The method used by the queue processor to write data to the printer depends on the data type:

- PM_Q_STD Data is written to the printer using the API:

Component	Action
Queue processor	Opens a DC for the printer using DevOpenDC and enables it as an OD_DIRECT device for PM_Q_STD data.
QP	Issues Escape(DEVESC_STARTDOC).
QP	Writes data, which is in metafile format, to DC.
Driver	Starts a journal file.
Driver	Processes and journals each call as it is passed by the queue processor.
QP	Issues Escape(DEVESC_ENDDOC).

© Copyright IBM Corp. 1990

Driver	Stops the journal file.
Driver	Plays each band to the device using the Prt... interface.
QP	Closes printer DC using DevCloseDC.
Driver	Closes printer using PrtClose.

- PM_Q_RAW

Component	Action
Queue processor	Opens a DC for the printer using DevOpenDC and enables it as an OD_DIRECT device for PM_Q_RAW data.
QP	Issues Escape(DEVESC_STARTDOC).
Driver	Opens the device for printing, using PrtOpen.
QP	Writes the data to the DC.
Driver	Sends the data to the device without further processing. The data is buffered into blocks of convenient size and written to the printer using PrtWrite. Control codes are sent to the printer using the PrtDevIOCtl commands. The size of the block depends on the amount of memory available to the device driver.
QP	Issues Escape(DEVESC_ENDDOC).
QP	Closes printer DC using DevCloseDC.
Driver	Closes printer using PrtClose.

- User Data Types

 The processing required for user data types depends on the format of the data type. In some instances it may be necessary to create a special queue processor to support the data type.

SplQpPrint

On page 17-26, add the following note to the description:

Note: SplQpPrint can be interrupted by an asynchronous call, on another thread, at entry SplQpControl.

SplQpQueryDt

On page 17-27, change the description of the second parameter (**paszDatatypes**) to:

Pointer to an array of pointers that address the locations for the returned names. Each location should be an array of 16 characters to accommodate the name and its terminating null.

Master Index

© Copyright IBM Corp. 1990

Index

C

category 1, function 53H 20
category 1, function 73H 20
category 4, function 50H 20
category 4, function 56H 20
category 5, function 4DH 21
category 5, function 4EH 21
category 5, function 6EH 22
character device data stream 25
character monitors 25

D

data stream monitoring 25
dependent device driver 11
DevHlp services 7
DevHlp, MonWrite 8
device data streams 25
device monitor limitations 25
device-dependent device driver 14
device-independent device driver 11
Disable_Device request 15
Disable_Support request 13
DispMsg 7
DosMonReg 15

E

Enable_Device request 15
error codes, field status 5

F

font monitor packet 16

G

GreCreateLogColorTable 29
GreDeviceSetAVIOFont 31
GreDeviceSetDCOrigin 31
GreEscape, DEVESC_DRAFTMODE 30
GreEscape, DEVESC_GETCP 29
GreEscape, DEVESC_SETMODE 29
GreGetAttributes 35
GreQueryLogColorTable 29
GreSetupDC 33

I

IDC interface descriptions 11
INIT-Time function call summary 3
IOCtl commands
 category 1, function 53H 20
 category 1, function 73H 20
IOCtl commands *(continued)*
 category 4, function 50H 20
 category 4, function 56H 20
 category 5, function 4DH 21
 category 5, function 4EH 21
 category 5, function 6EH 22
 category 8, function 43H 23
 category 8, function 63H 23

J

job title packet 16

K

keystroke monitors 15

M

monitor dispatcher interface 16
monitor-job title packet 17
monitoring character device data streams 25
monitors
 character 25
 device 25
 keystroke 15
 print 16
MonWrite, DevHlp 8
mouse device 11
MOUSE$ 11

P

pointing device support 11
print-monitor-job title packet 16
printer timeout value 22
Process_Absolute request 13
Process_Packet request 12

Q

Query_Config request 14
queue processor 37

R

Read_Disable request 15
Read_Enable request 14
request packets 5
requesting an IDC interface 11
return printer timeout value IOCtl 22

© Copyright IBM Corp. 1990

S

set print job title IOCtl 21
set printer timeout value IOCtl 21
SplQpPrint 38
SplQpQueryDt 38

V

VideoPause 9

IBM United Kingdom
International Products Limited
PO Box 41, North Harbour
Portsmouth, PO6 3AU
England

Printed in Denmark
Scanprint a-s, Viby J

